

SYRUP

LLMの仕組み

海辺 康志

2026/09/19

イントロダクション

- ChatGPTやClaudeのようなAIは、**LLM（大規模言語モデル）**という技術を使っている
- LLMを使いこなすには、LLMが内部でどう動いているかを知っておく事が役に立つ
- 今日のゴール
 - 1. 入力から出力までの「生成の流れ」をつかむ
 - 2. なぜ「指示の与え方（コンテキスト）」が結果を左右するのかを理解する
- 数式は最小限に、仕組みのイメージを掴んでいく

1. LLMの概要

2. Transformerの誕生

3. Attention（注意機構）

4. トークン化・Embedding

5. Self-AttentionとQVK

6. Multi-Head Attention

7. Transformer Blockの全体像

8. パラメータ・学習

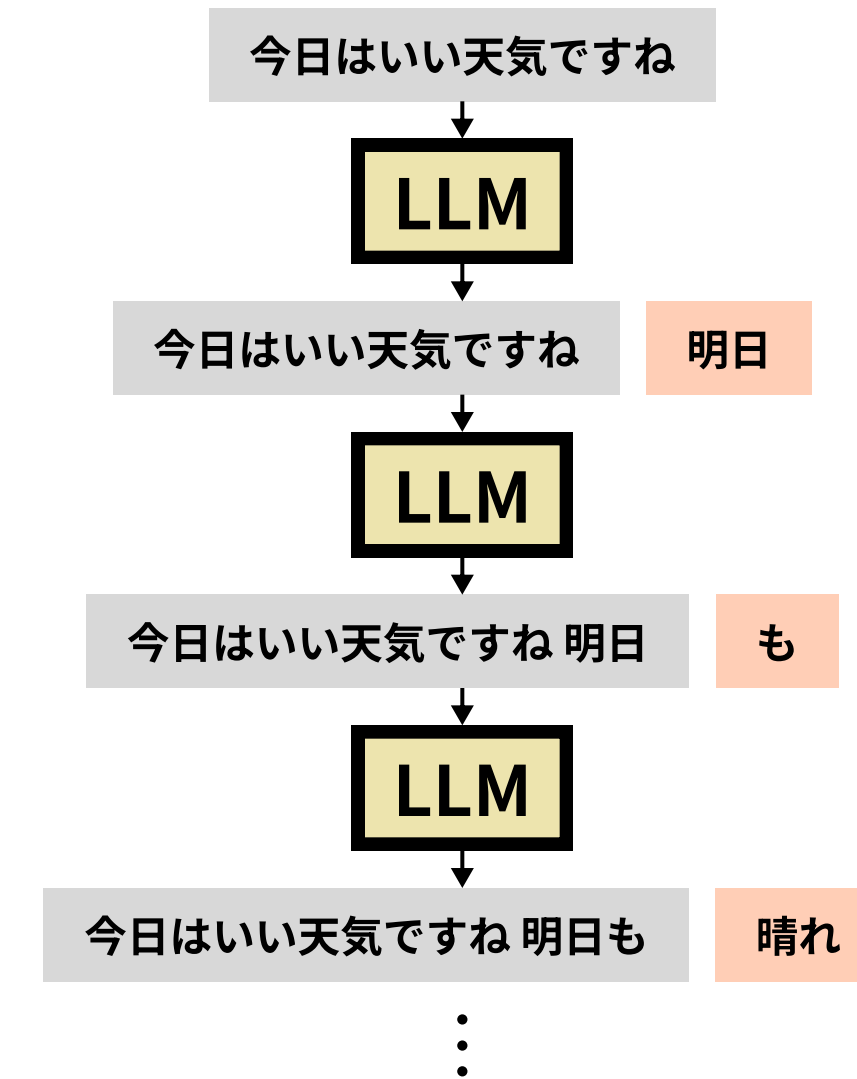
9. なぜ「推論」できるように見えるのか

10. LLMの限界

11. コンテキストエンジニアリングへ

LLMの全体像

- LLMがやっていることをひとことと言うと：**次に来る単語（トークン）を予測する**
- 大まかな流れ
 1. 文章を細かい単位（トークン）に分割
 2. それぞれを数値のベクトルに変換
 3. Transformerという仕組みで単語同士の関係を調べる
 4. 「次に来る単語」の確率を計算
 5. 選んだ単語を追加して、1に戻る（これを繰り返す）
- つまりLLMは「文章を丸ごと理解(解析)」しつつ、「次に来る単語を連続で予想」しているだけ



関連用語

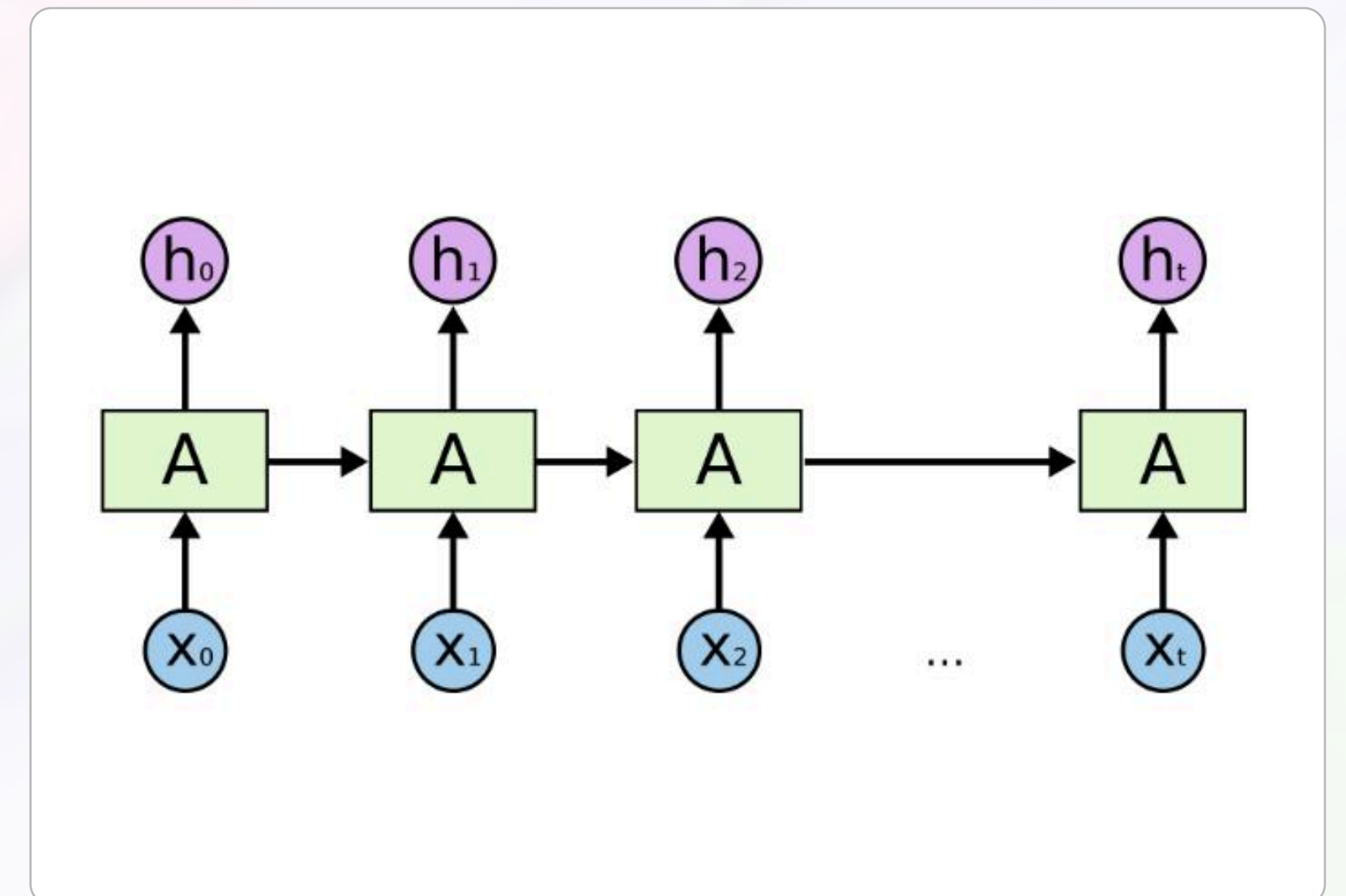
トークン

ベクトル

Transformer

AIの文章理解の歴史

- 以前のAIは、文章をどう読んでいたか？
- Transformer以前の主流は**RNN（再帰型ニューラルネットワーク）**
- RNNは文章を**先頭から順番**に、1語ずつ処理していく
- 弱点
 - 長い文章になるほど、最初の方の情報を忘れやすい
 - 1語ずつしか処理できないため、計算を並列化しにくく、学習に時間がかかる
- 例：「太郎は公園でボールを拾って、**彼**はそれを犬に投げた」→「**彼**」が何を指すか、離れた単語を覚えていないと分からない



参照: https://web.stanford.edu/class/cs379c/archive/2018/class_messages_listing/content/Artificial_Neural_Network_Technology_Tutorials/OlahLSTM-NEURAL-NETWORK-TUTORIAL-15.pdf

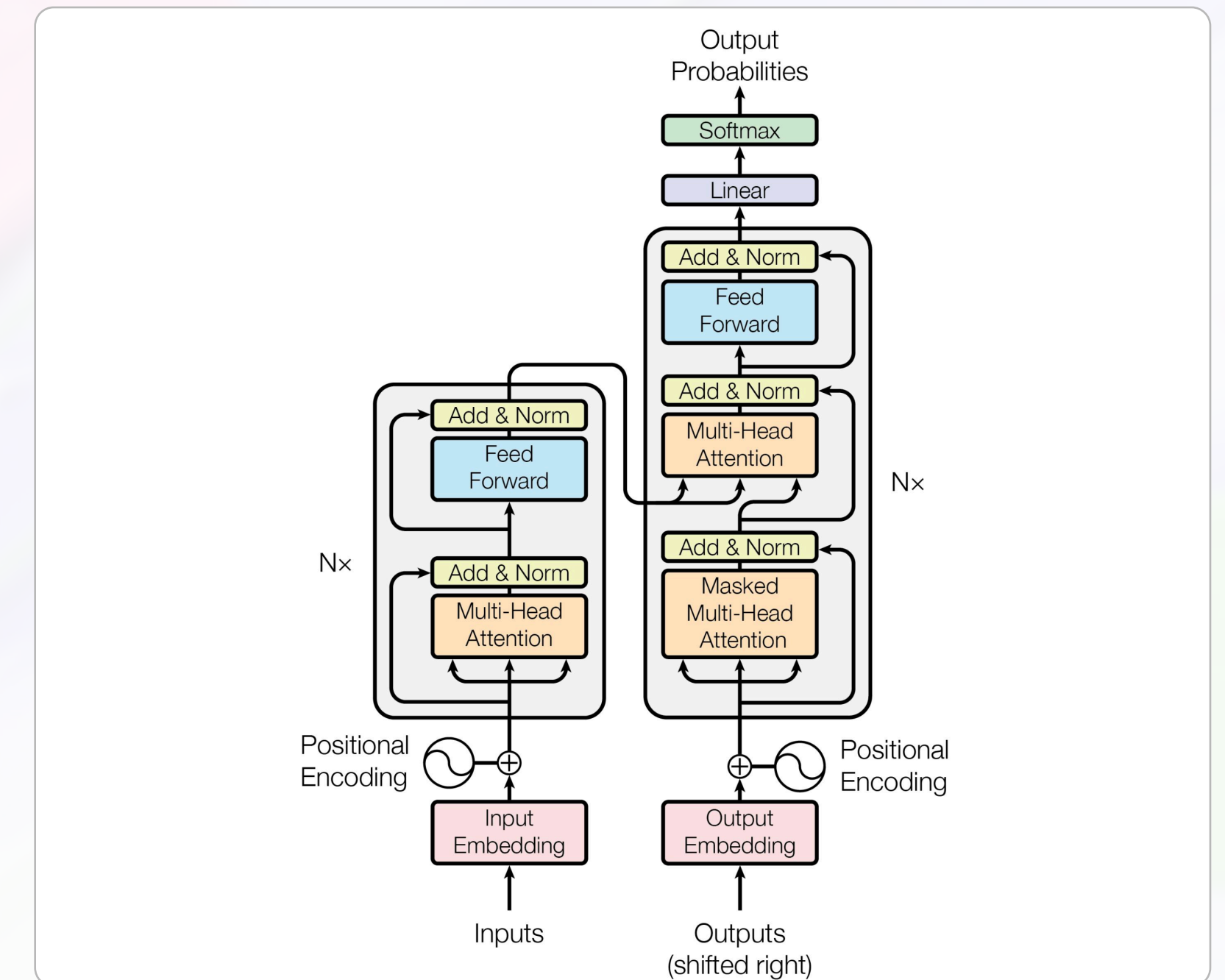
関連用語

RNN

並列処理

Transformerのはじまり

- 転換点になったのは、2017年に発表された1本の論文
- Googleの研究者たちが「**Attention Is All You Need**」という論文を発表
- タイトルの意味は「**注意機構（Attention）だけで十分**」
- この論文が提案した**Transformer**という構造が、今のLLMの土台になっている
- ChatGPTのGPTは「**Generative Pre-trained Transformer**」の略。名前にTransformerが入っている



参照: <https://arxiv.org/pdf/1706.03762>

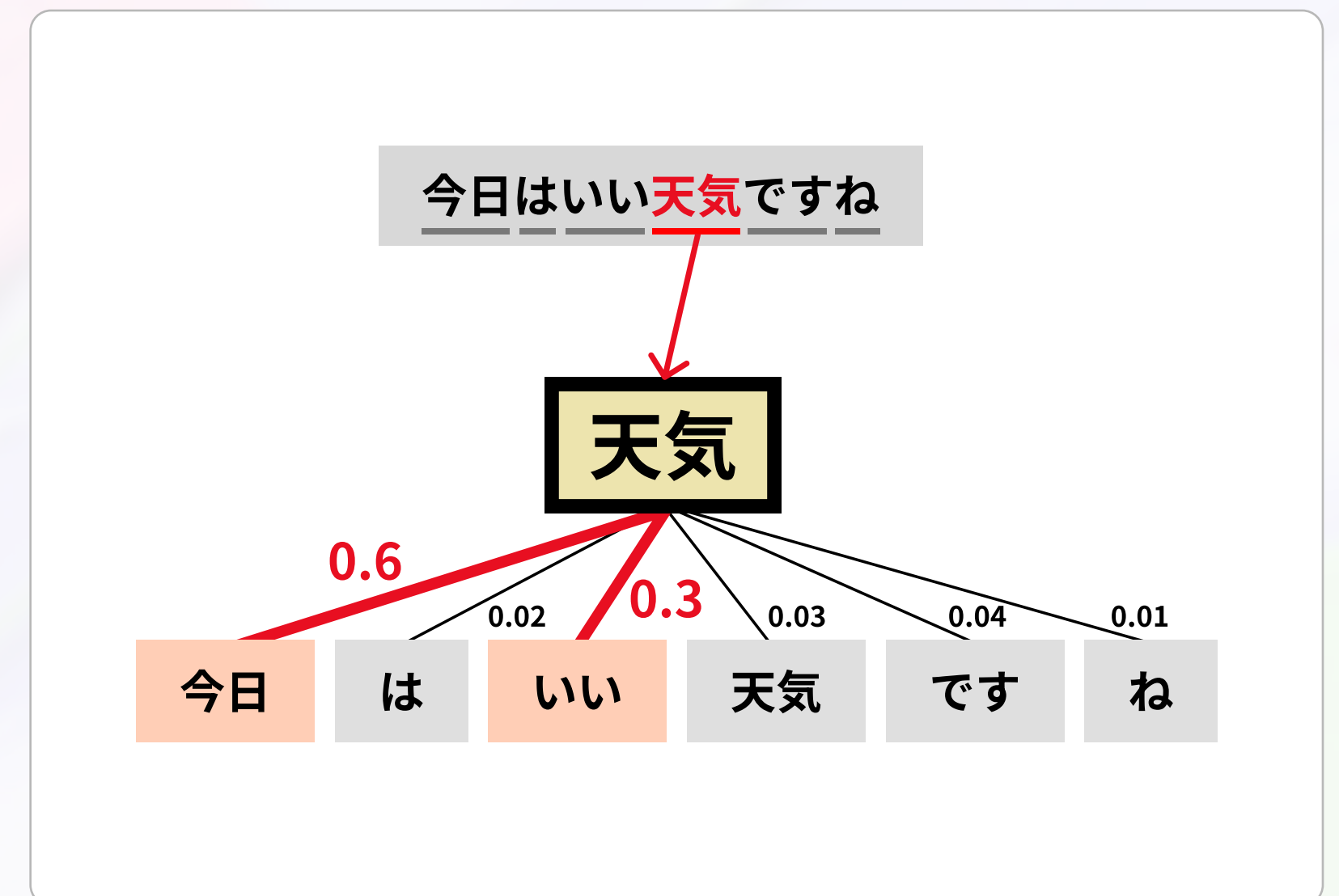
関連用語

Transformer

GPT

Attention（注意機構）

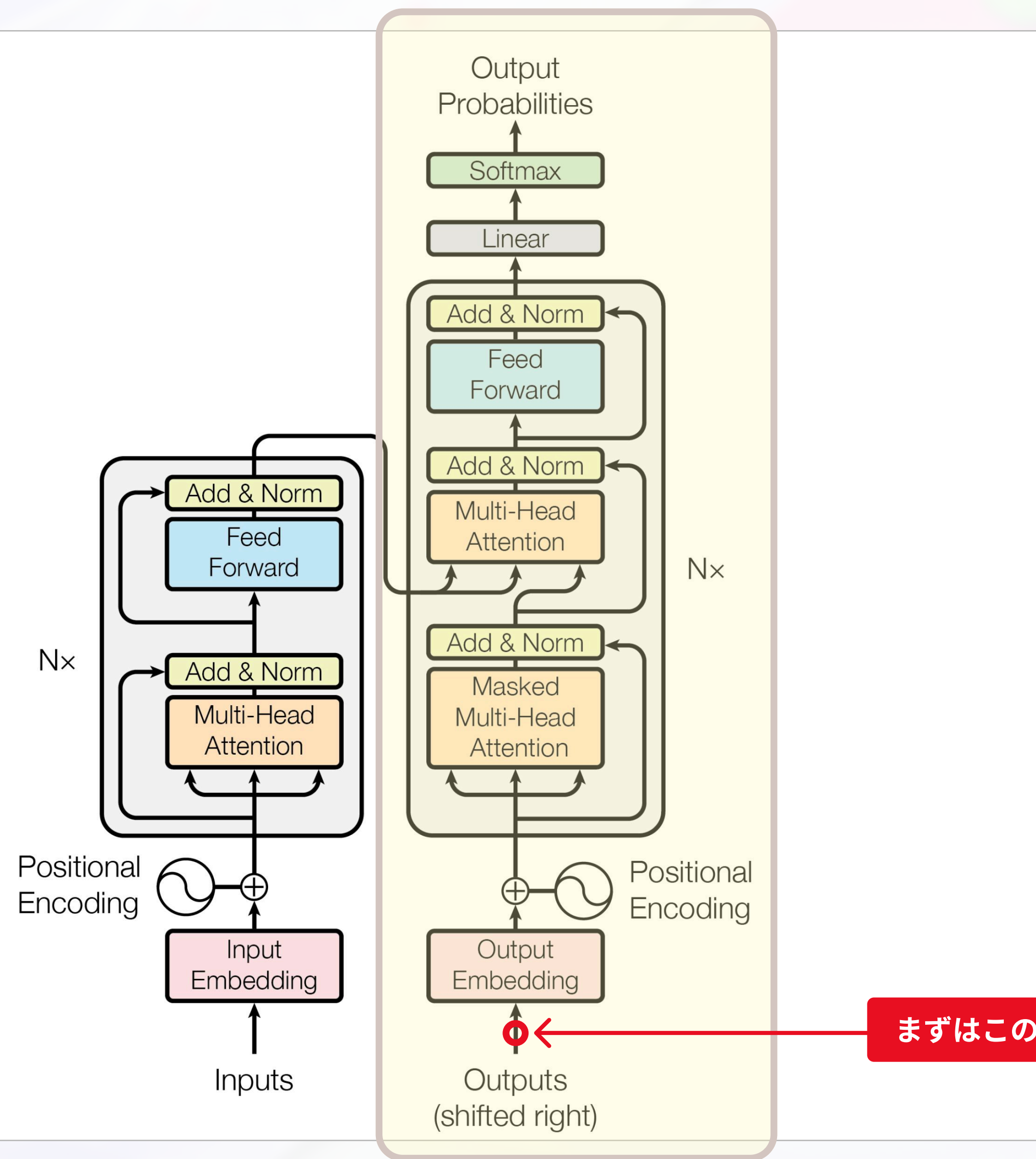
- Transformerの核心は**Attention（注意機構）**
- 文章の中の単語すべてを同時に見て、関連の強い単語同士を結びつける仕組み
- 学習時に順番に読む必要がなく、**全単語を同時（並列に）計算する**
- 学習が速くなり、大量データを扱える
- LLM誕生のきっかけとなる



関連用語

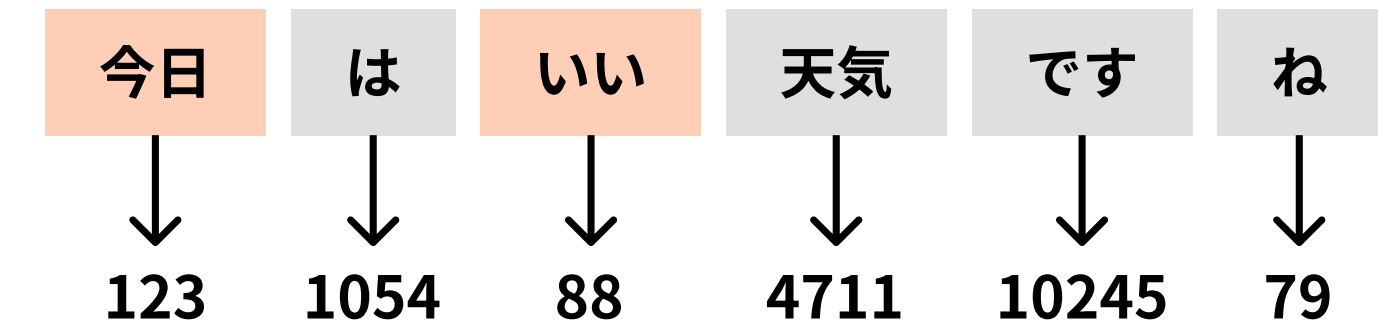
Attention

**ここからはTransformerの
内部を紐解いてゆきます**



トークン化

- Transformerの処理をする前段階
- まず文章をAIが理解できる**トークン**という小さな単位に分割する
- トークンは「単語」とは限らない。「学習」「する」のように、単語の一部や活用ごとに分かれることも
- 同じ文章でも、使うモデルやトークナイザーによって分割のされ方は変わる
- 英語は単語単位に近く分割されやすいが、日本語は文字や部分文字列で分かれることが多い
- 分割された各トークンには、あらかじめ決まったID（番号）が割り振られる

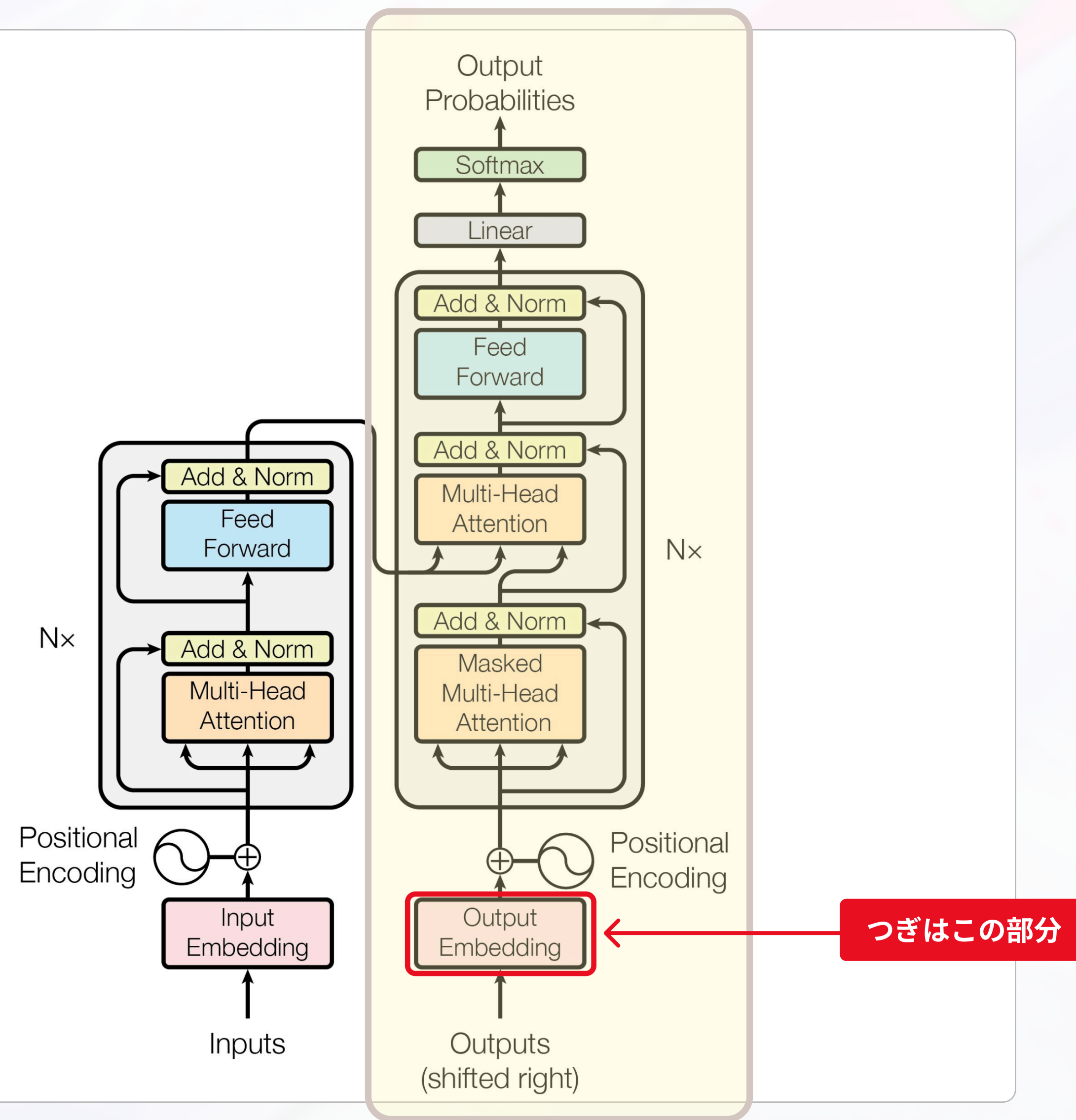


文章を分割し数字(ID)に変換

関連用語

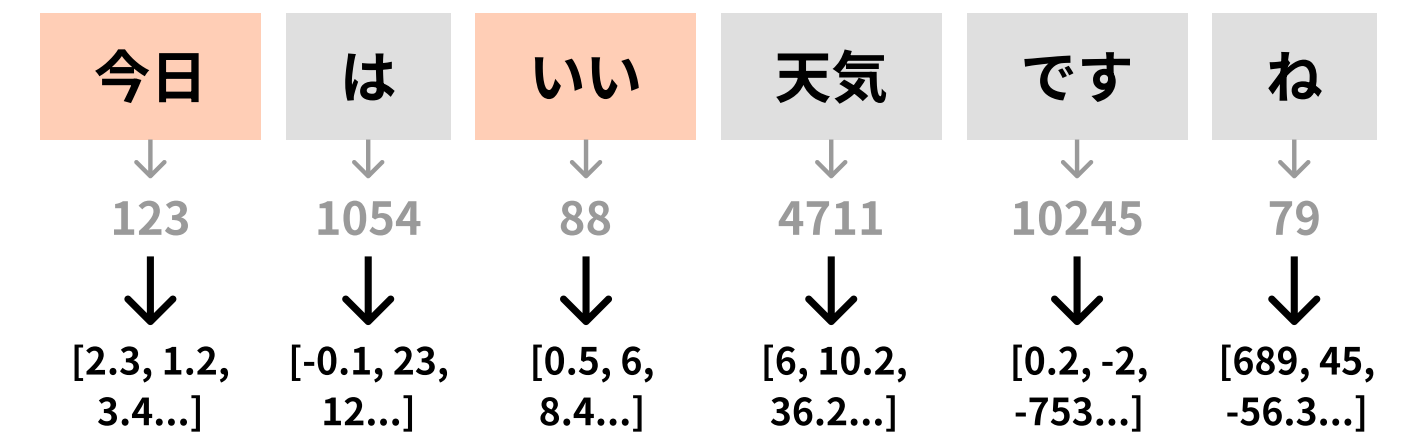
トークン

トークナイザー



Embedding

- トークンのIDはただの番号なので、そのままでは「意味」を表せない
- そこで各トークンを、**Embedding（埋め込み）**という処理で「意味を表す数百～数千次元のベクトル（座標）」に変換
- 例：「王様」と「女王様」は**近い場所**の座標、「王様」と「バナナ」は**遠い場所**の座標に
- この座標が散らばった空間上で、単語同士の関連度を計算できるようになる



各トークンをベクトルに変換

関連用語

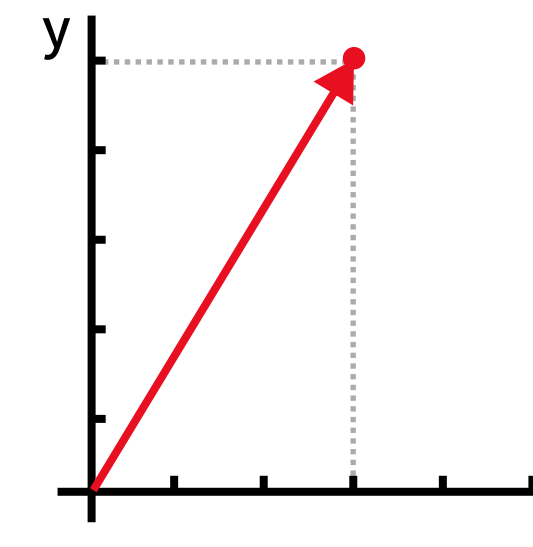
Embedding

ベクトル

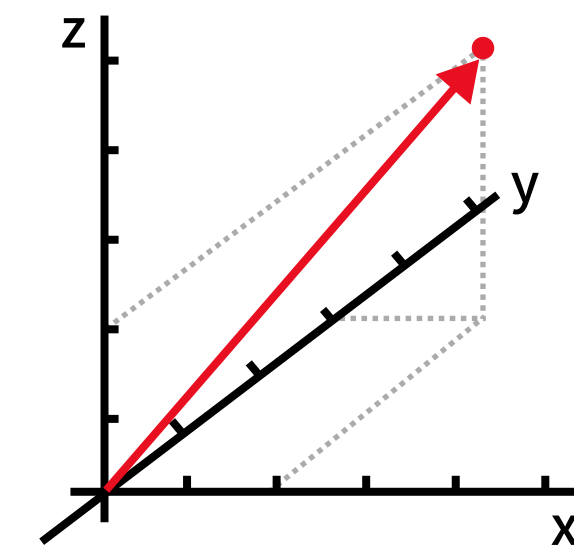
ベクトルとは（補足）

- **ベクトル**とは、複数の数値を順番に並べたもの。例えば **(3, 5)** のように、数字の組がそのまま2次元ベクトルになる
- グラフの上では、ベクトルは「**点の位置と方向**」を表す
- 要素の個数（**次元**）が増えても考え方は同じ。3個なら3次元、100個なら100次元になり、より**多くの特徴を同時に表現できる**ようになる
- LLMでは「**言葉の意味**」としてベクトルを使用する。数百～数千個の数字で1つの単語の意味を表現する

(3, 5)



(2, 3, 2)



関連用語

ベクトル

次元

ベクトル同士の類似度：内積（補足）

- ベクトル同士を比べる方法の1つが「**内積**」
- 計算は**対応する数字同士を掛けて、全部足す**だけ
- 例：(2, 3)と(4, 1)の内積 → $2 \times 4 + 3 \times 1 = 11$
- 内積の値が大きいほど、2つのベクトルは「似ている」**
- 逆方向を向いているとマイナスに、直角に近いとほぼ0に**
- この「**掛けて足すだけで、類似度計算ができる**」という点がGPUでの並列計算に向いている

○なす角を使う方法

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \underline{\cos\theta}$$

方向の違いがわかる

○要素を使う方法

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \underline{a_1 b_1 + a_2 b_2 \cdots + a_n b_n}$$

計算が単純でGPUでの並列計算に向いている



内積の値が大きい



内積の値が0

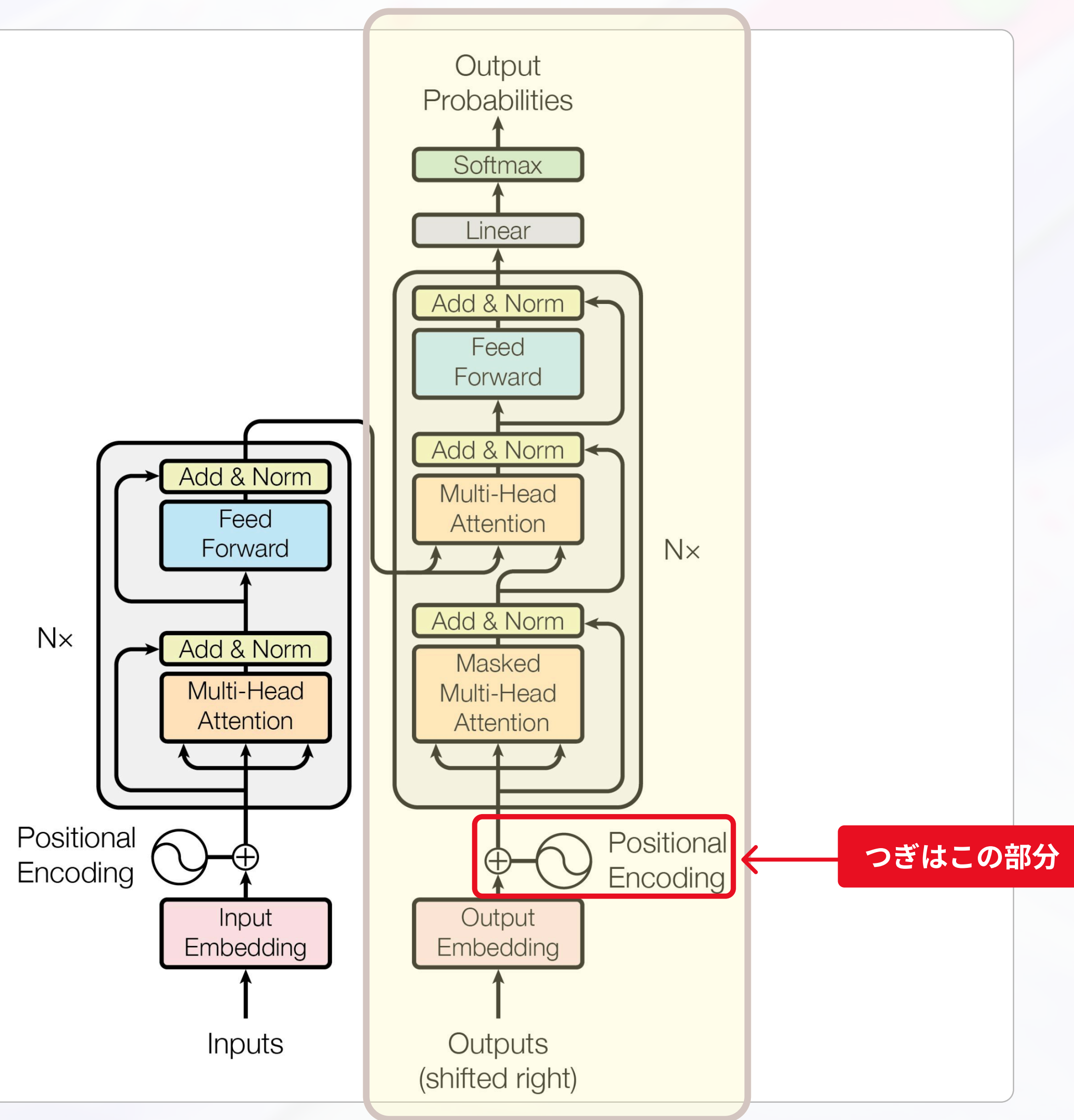


内積の値はマイナスに

関連用語

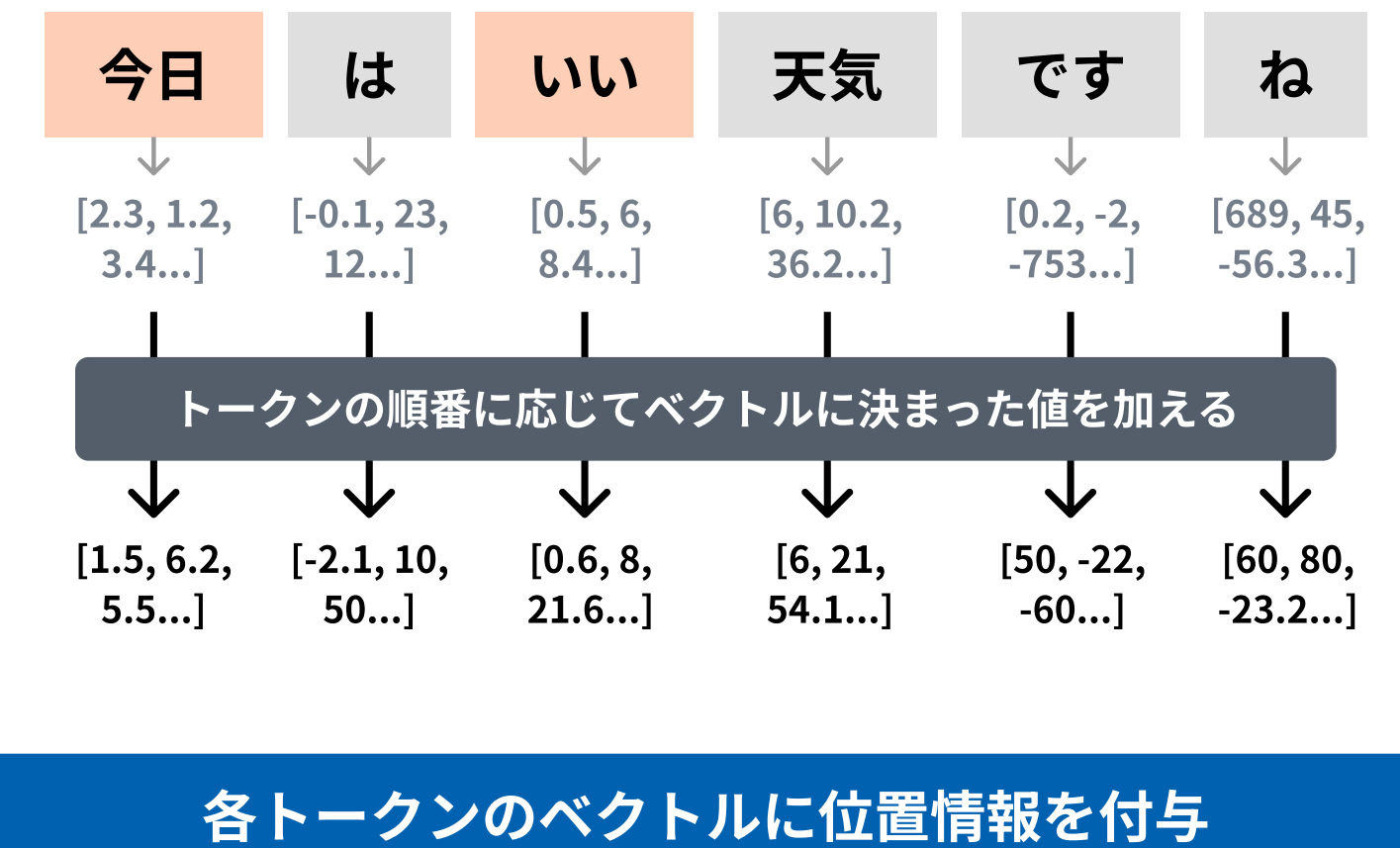
内積

線形代数



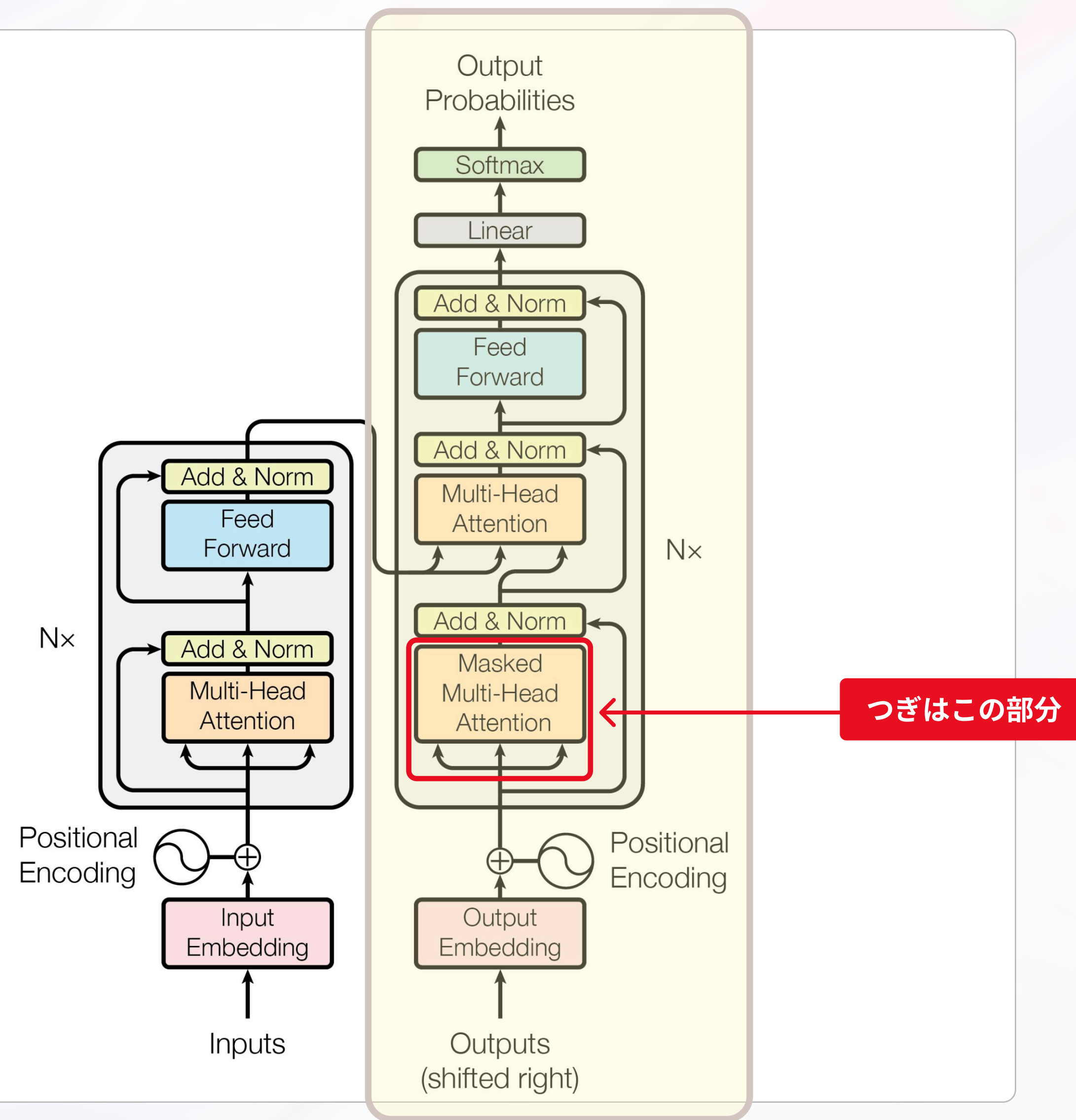
位置情報

- Attentionは単語を並列に（同時に）処理するため、そのままでは「どの単語が何番目にあるか」という順序の情報が失われる
- 「猫が犬を追いかけた」と「犬が猫を追いかけた」は出現する単語な同じでも意味が逆になるので、位置情報が必要
- そこでTransformerは**Positional Encoding（位置エンコーディング）**という仕組みで、各トークンのベクトルに「これは何番目の単語か」という情報を追加する



関連用語

Positional Encoding



Self-Attention

- ここからが「Attention」の本体：Self-Attention
- 文章内の各単語が、同じ文章内の他のすべての単語と「どれくらい関係が強い」かを計算する仕組み
- 例：「彼女はギターを弾いた。彼女は音楽が好きだ」
- 2つ目の「彼女」というトークンが、文中の他のすべての単語（「ギター」「弾いた」など）との関連度を計算し、最初の「彼女」との関連が特に強いと判断する
- この「関連度」を数値化する仕組みが、次のスライドで説明する $Q \cdot K \cdot V$

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

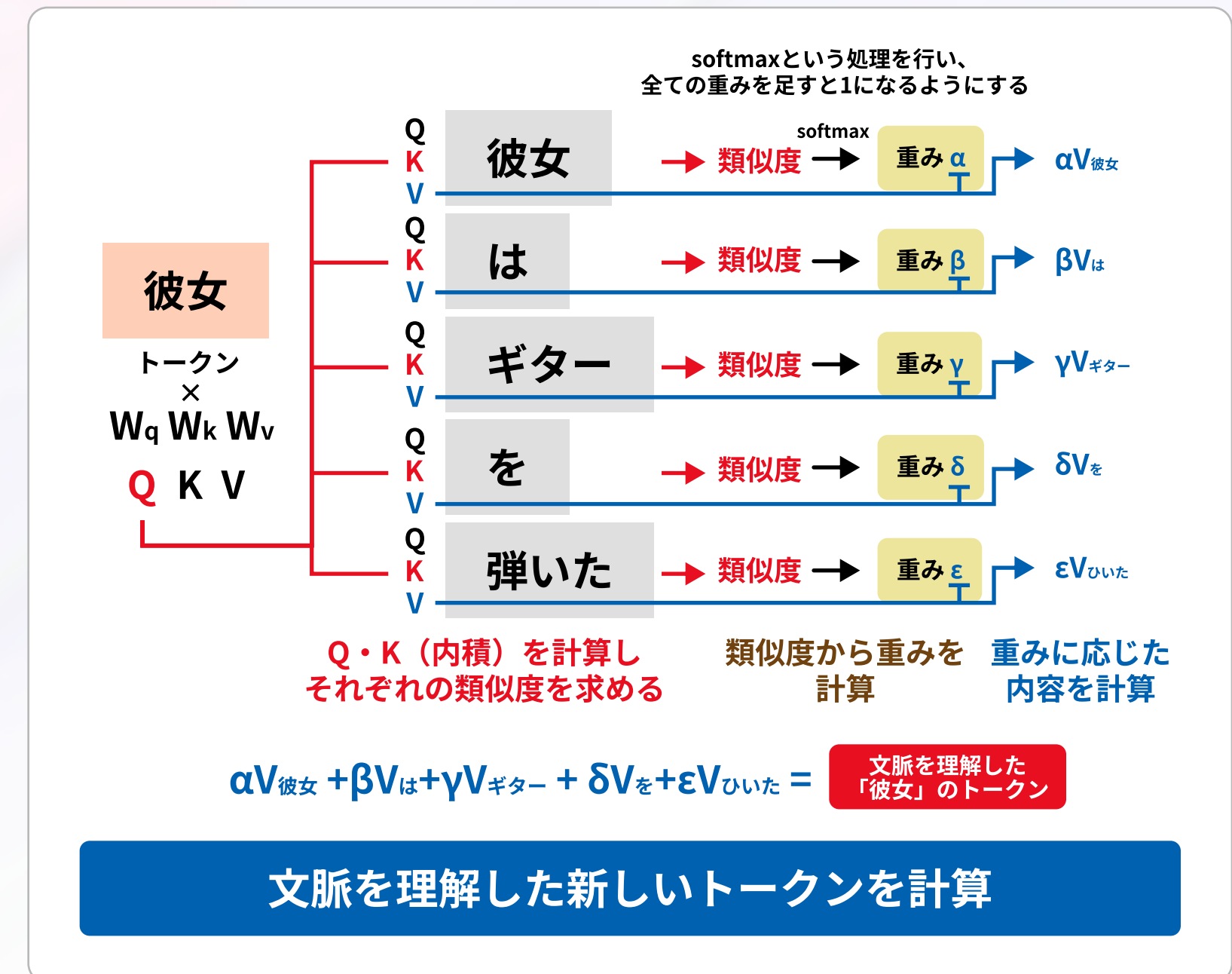


Q・K・V

- 内部では、各トークンが3種類のベクトルに分解される
Query（検索クエリ）：「自分が何を探しているか」
Key（見出し）：「自分の情報のキーワード」
Value（中身）：「自分の情報全部」
- 図書館で例えると
Query＝探したい検索キーワード
Key＝自分の本のキーワード
Value＝自分の本の中身そのもの
- 自分のトークンのQueryと、他の全トークンのKeyを見比べて「どれだけ似ているか」を計算 → その一致度に応じて、各トークンのValueを重み付けして混ぜ合わせる
- この「重み付けして混ぜ合わせた結果」が、そのトークンの新しい（文脈を反映した）表現になる

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

$\sqrt{d_k}$ ：スコアが大きくなりすぎないように調整するための項



関連用語

Query

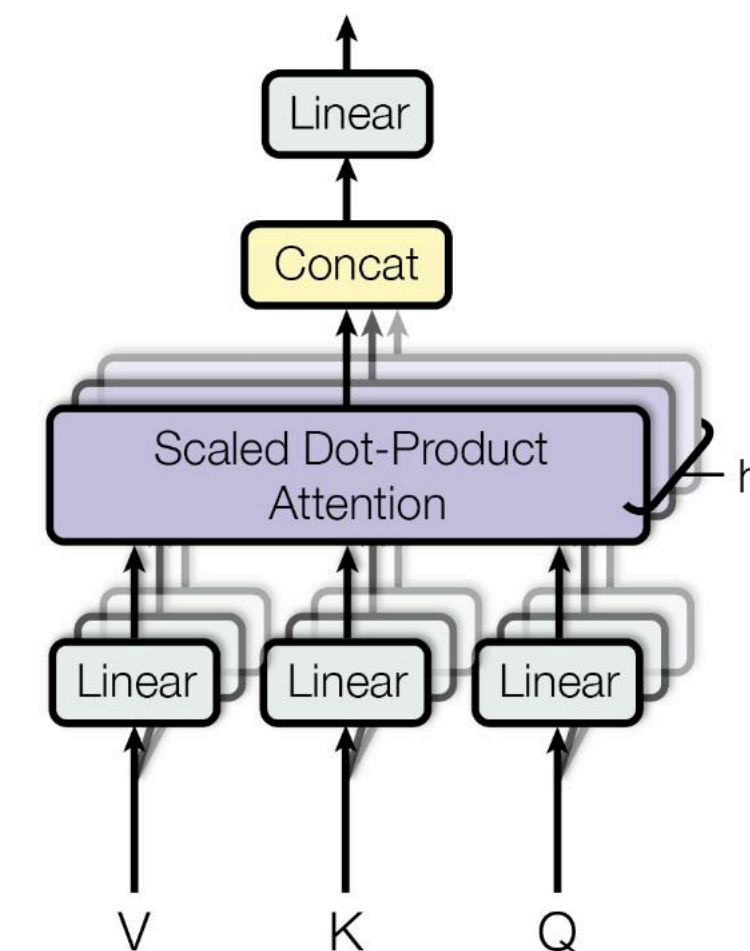
Key

Value

softmax

Multi-Head Attention

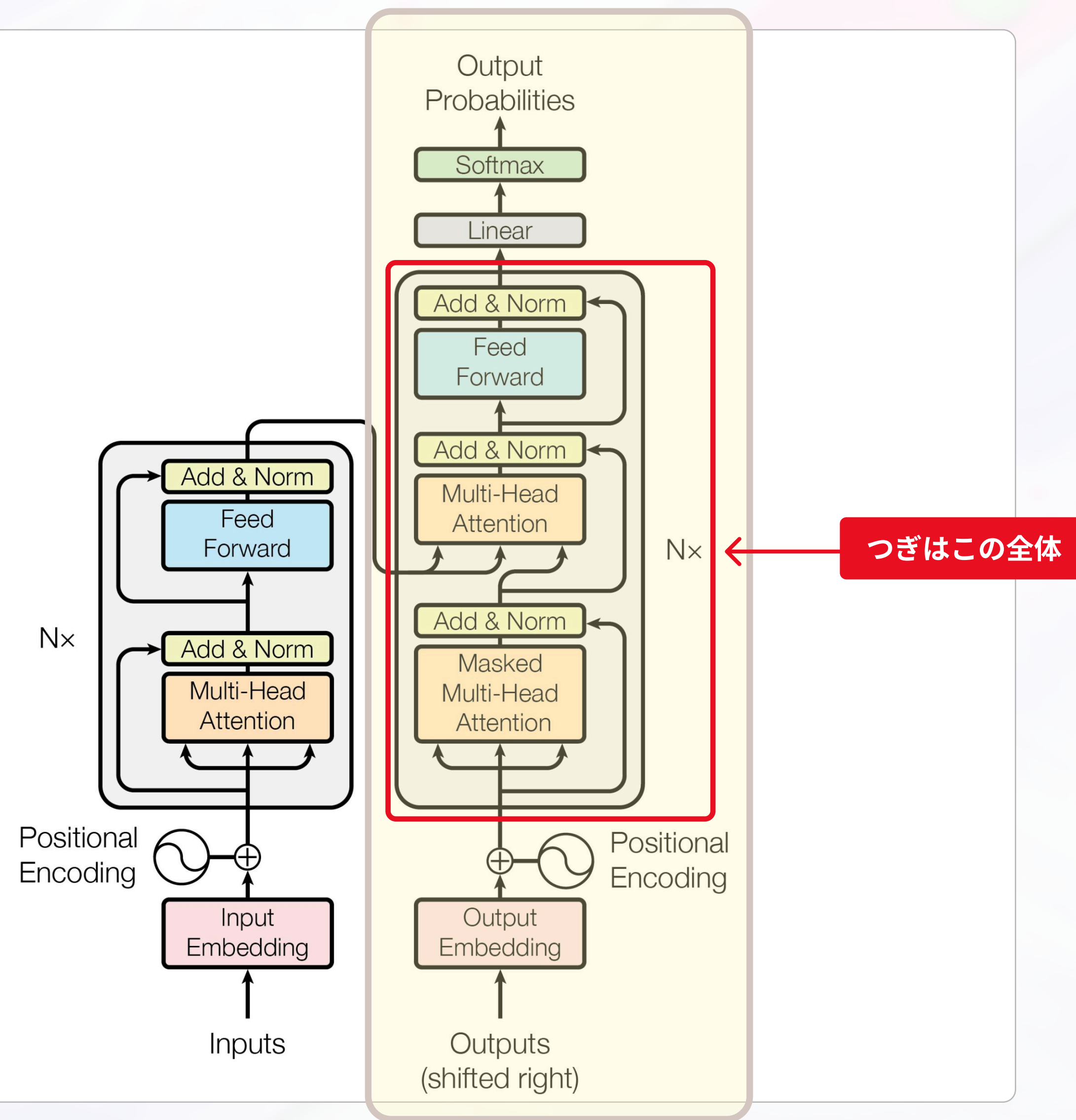
- 1回のAttention計算では、1つの観点でしか単語の関係を見られない
- そこでTransformerは、**QKVの計算を複数の「Head」で並行して行う**
- 例：あるHeadは「文法的な関係」に注目し、別の頭は「意味の近さ」に注目する、というように役割が自然に分かれていく
- 複数の視点の結果を最後に1つにまとめることで、より豊かに文脈を捉えられるようになる



複数のヘッドでattentionを行い最後に統合

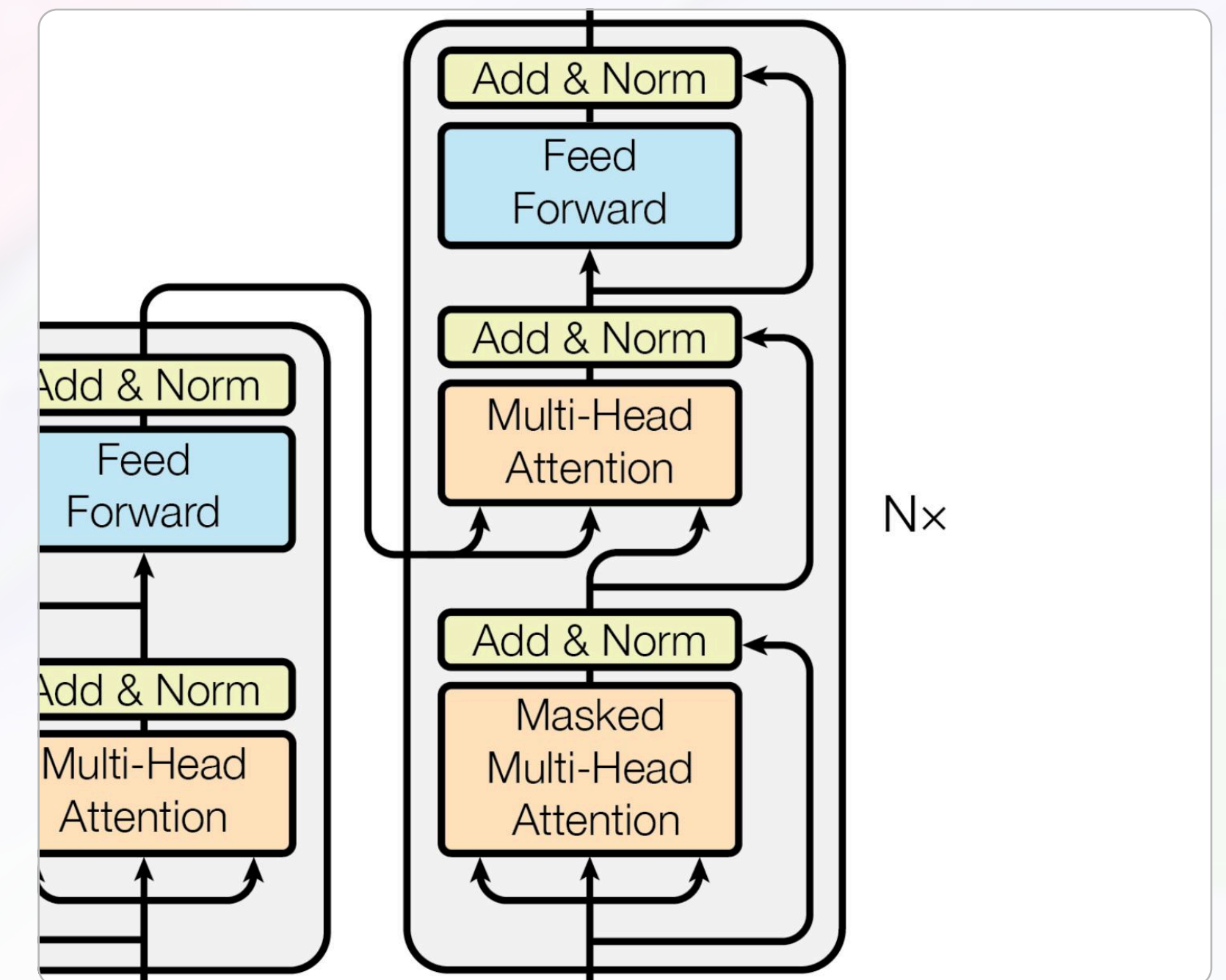
関連用語

Multi-Head Attention



Transformer Blockの全体像

- Multi-Head Attentionの出力は、そのまま次に渡されるわけではなく、いくつかの処理を経て**Transformer Block**という1つの塊にまとまっている
- 元の論文 ("Attention Is All You Need") での並び順
 1. Multi-Head Attention
 2. **Add & Norm** (残差接続 + Layer Normalization)
 3. **Feed Forward Network** (FFN)
 4. Add & Norm (もう一度)
- このBlockを**何十層も積み重ねる**ことで、単語同士の関係をどんどん深く・複雑に読み取れるようになる



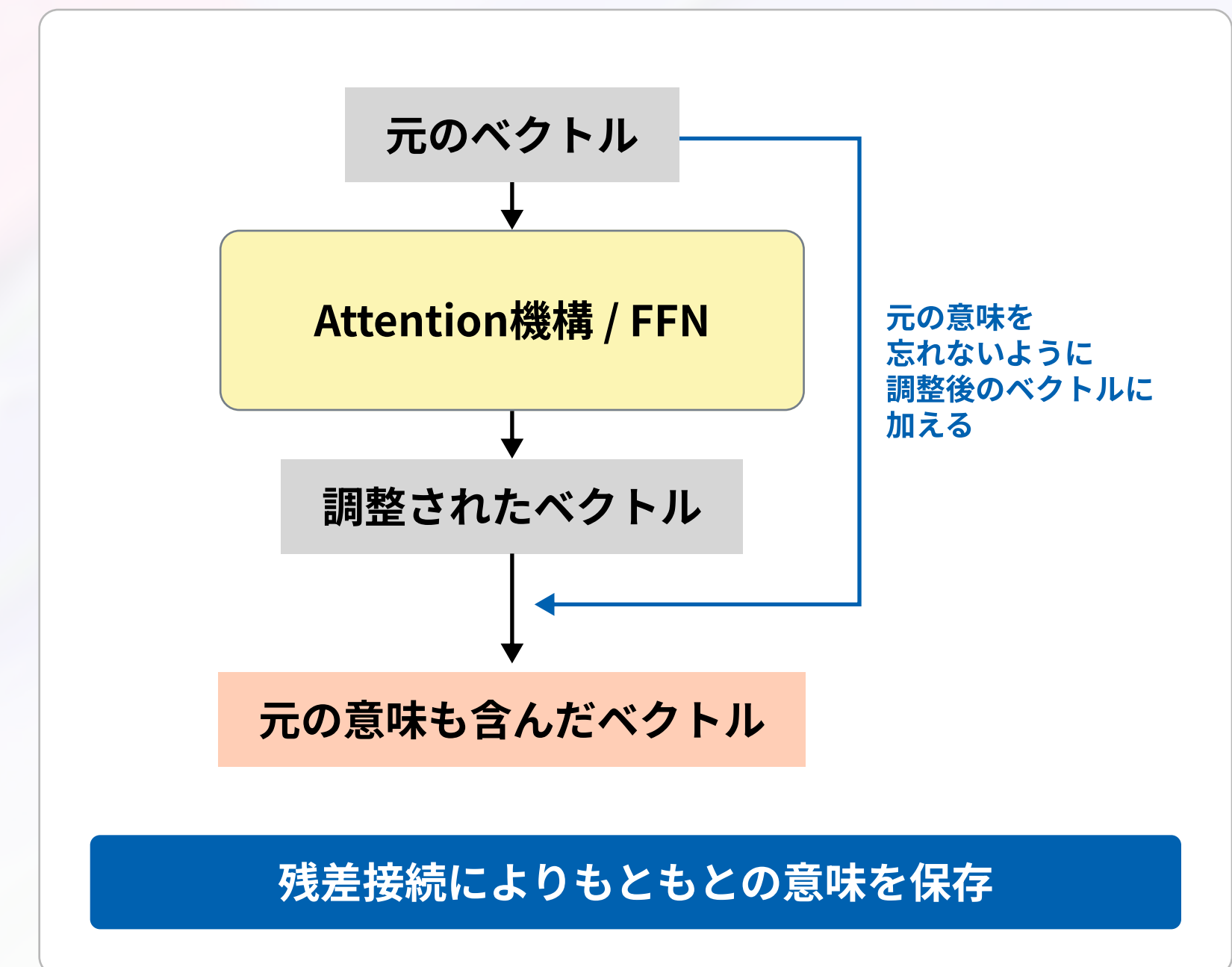
関連用語

Add & Norm

Feed Forward Network

Add & Norm（残差接続とLayer Normalization）

- AttentionやFFNを通るたびに、元の情報が失われてしまわないか？
- **Residual Connection（残差接続）**：各層の「入力」を「出力」にそのまま足し合わせる仕組み
- 層を重ねても元の情報が消えず、深いネットワークでも学習が安定する
- **Layer Normalization**：各層の出力の値の大きさ（ばらつき）を整える仕組み
- 学習を安定させ、層を積み重ねても値が極端に大きく/小さくなりすぎないようにする
- この2つをセットで**Add & Norm**と呼び、AttentionとFFNのそれぞれの直後に配置される



関連用語

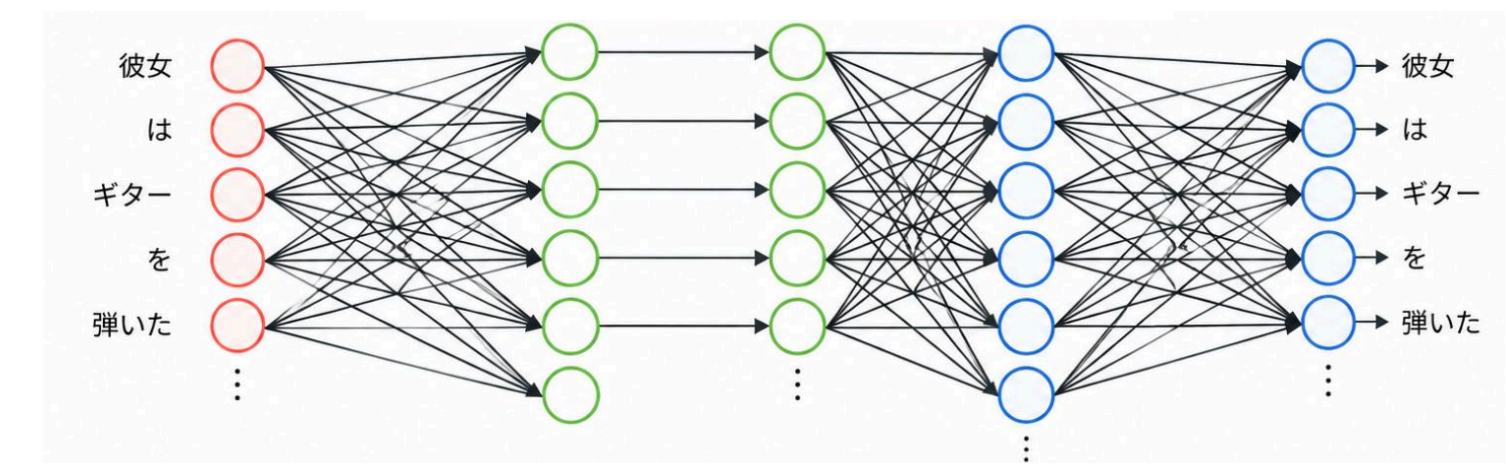
Residual Connection

Layer Normalization

Add & Norm

Feed Forward Network (FFN)

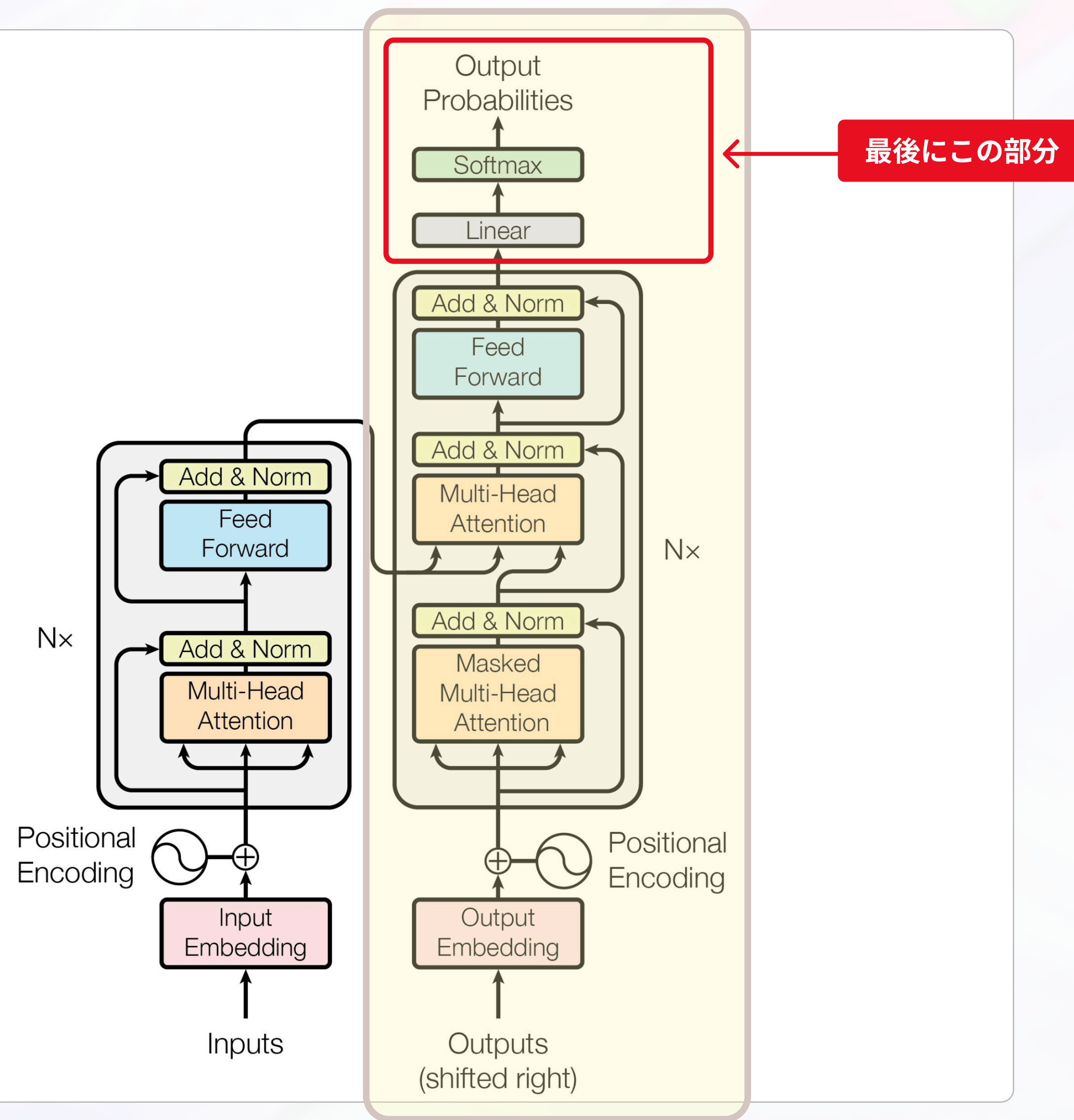
- **Feed Forward Network (FFN)** : 各トークンのベクトルを、1つずつ独立に整理する層
- 仕組みは単純：次元を一度広げて（線形変換）→非線形な関数を通す→また元の次元に圧縮する（線形変換）、という2段階
- Attentionのように「他のトークンを見る」ことはせず、それぞれのトークン単体に同じ加工を行う
- **Attentionが「関係」を扱う層だとすれば、FFNは「個々の情報の整理」を担う層、というイメージ**
- 近年の解釈可能性研究では、LLMが習得した知識の一部はこのFFN層に対応していると考えられている



Attentionで得た情報の特徴をひもとく層

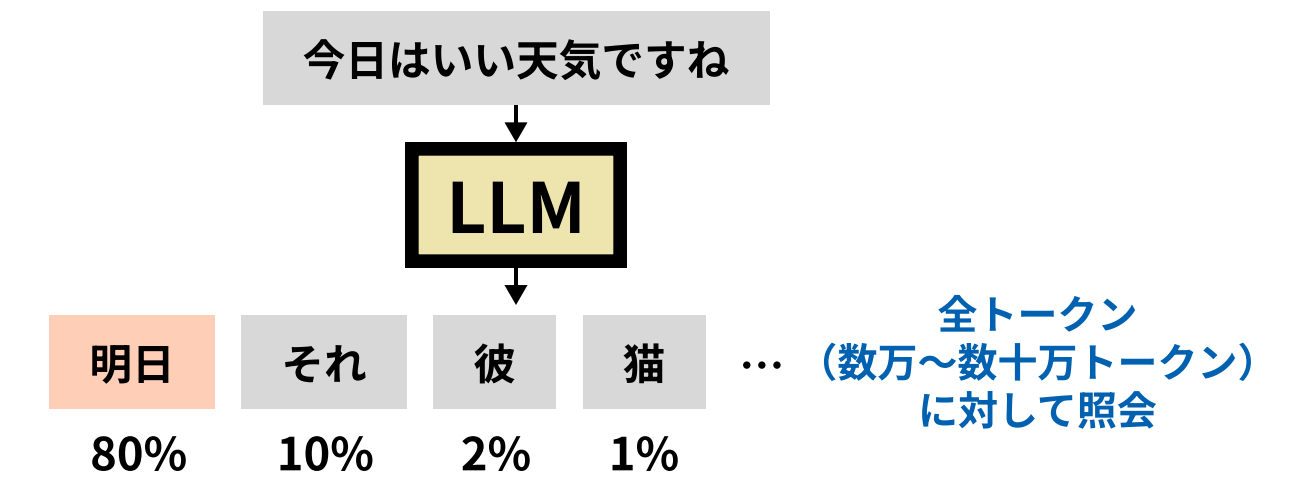
関連用語

Feed Forward Network



出力層と生成

- 何十層ものTransformer Blockを通過した「意味を表すベクトル」を次に来る単語の候補に変換するのが出力層
 - 1. **Linear層**：最終ベクトルを、語彙中の各トークンに対応するスコアへ変換する
 - 2. **Softmax関数**：そのスコアを「確率」に変換する
- 結果として「次のトークンは、"猫"である確率60%、"犬"である確率20%…」というような確率分布が得られる
- ここから実際に次のトークンを1つ選ぶ。選び方の調整に使われるのが**Temperature**（温度）というパラメータ
- **Temperatureを低くすると安定するが単調になりやすく、高くすると確率の差が小さくなり、多様な候補が選ばれやすくなるが不自然になりやすい**
- 選んだトークンを、それまでの文章の末尾に追加し、もう一度、最初から同じ処理を繰り返す
- これを1トークンずつ繰り返すことで、文章が生成されていく



最後のベクトルから次の単語を予想する

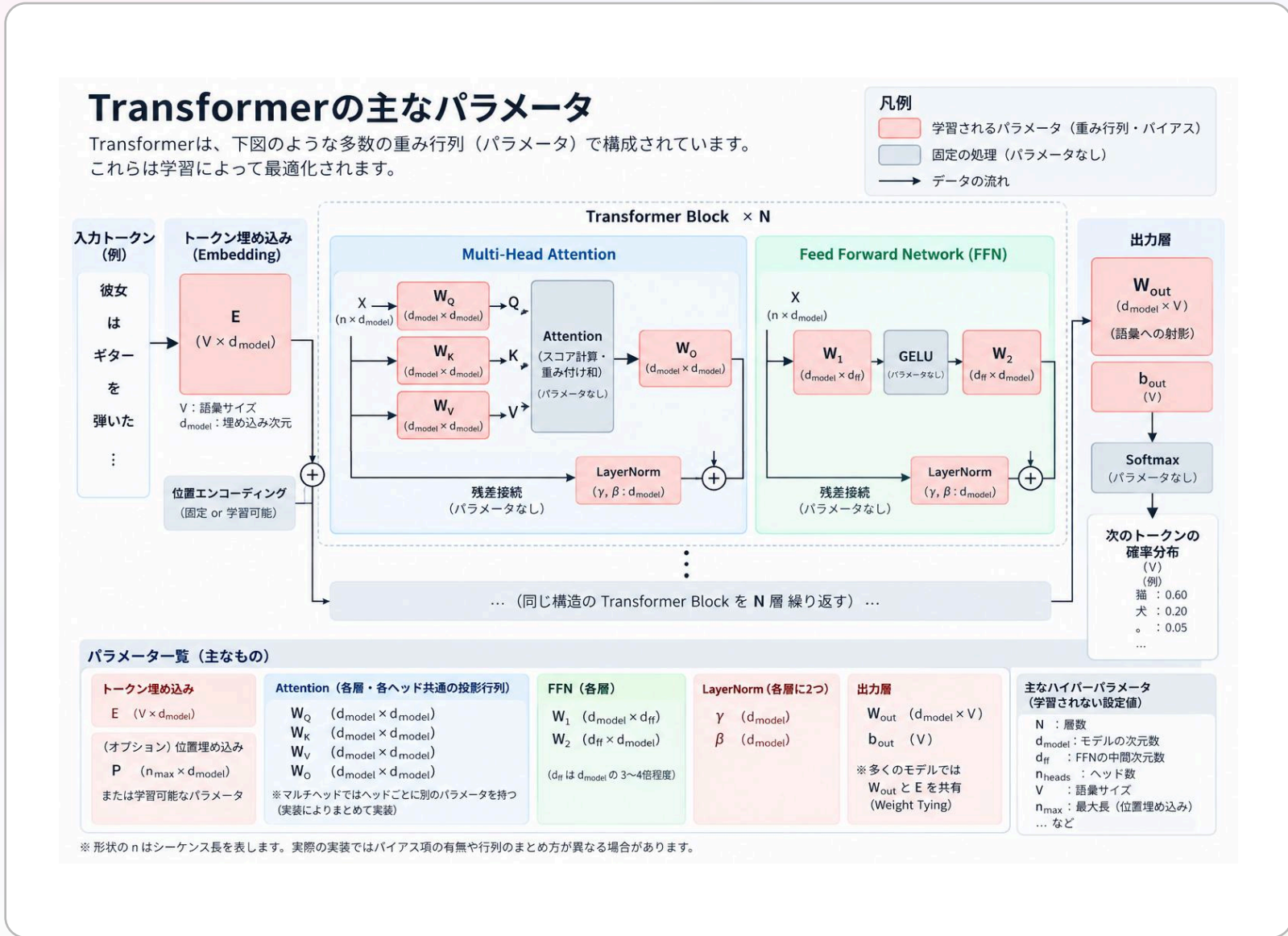
関連用語

Temperature

Transformerの内部が理解できたでしょうか

パラメータの正体：何を学習しているのか

- 実は、ここまでのスライドに登場した**行列の中の数字そのものが、すべてパラメータ**
- **Embedding表**：各トークンIDを意味ベクトルに変換する対応表
- **Q・K・Vへの変換行列**：各トークンのベクトルをQuery・Key・Valueに変換するための行列
- **FFNの行列**：各トークンのベクトルを加工する行列
- **出力層のLinear行列**：最後のベクトルを、語彙の各単語へのスコアに変換する行列
- これらの行列は、GPT-3クラスで数千億個、最新モデルではさらに多くの数値で構成される
- 学習とは、この**大量の行列の中の数字を、少しずつ書き換えていく**こと
- 一度学習が終われば、これらの数字は固定され、どんな入力に対しても同じ行列がそのまま使われる

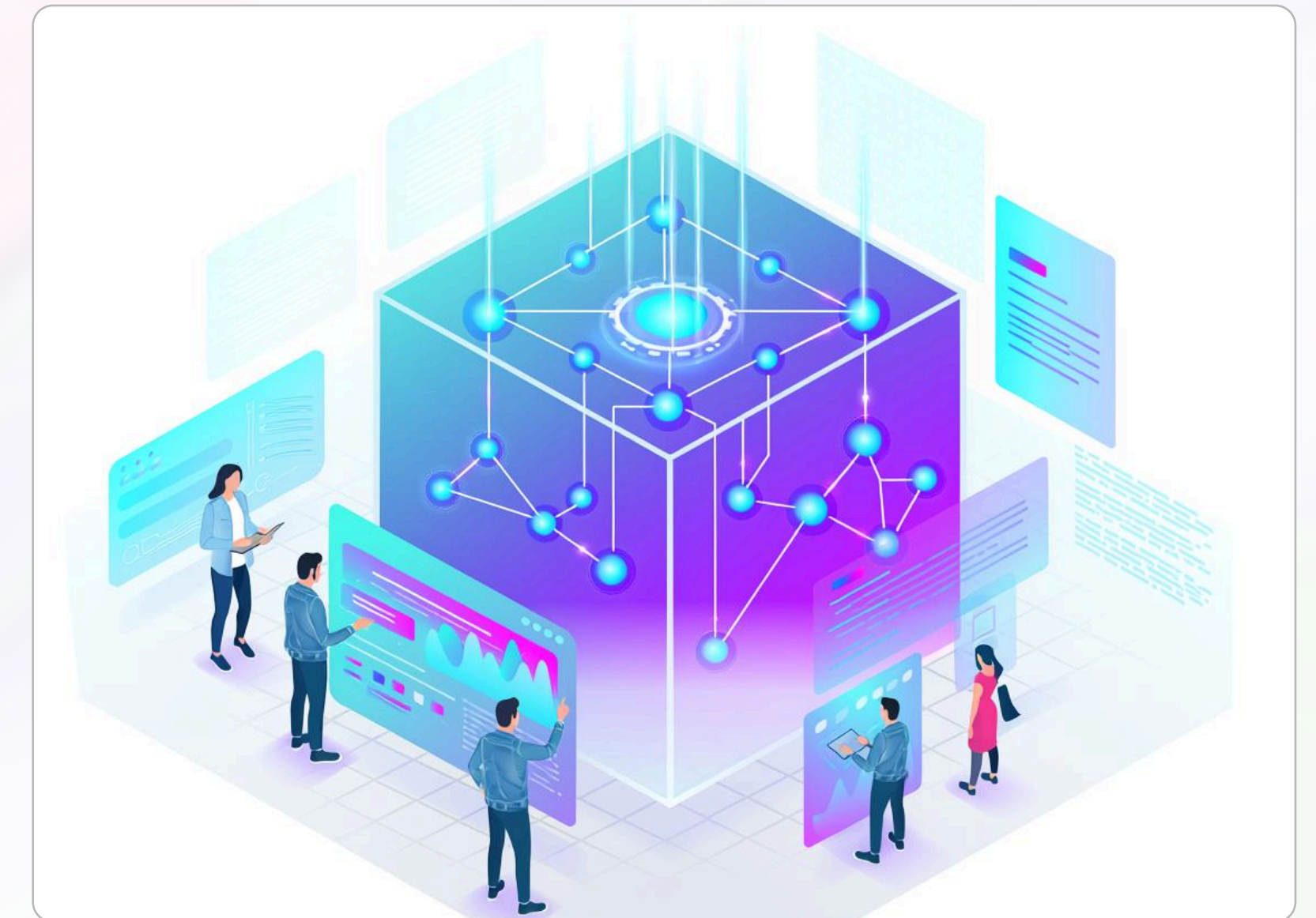


関連用語

パラメータ

学習

- ここまでは「学習済みのLLMが動く仕組み」。ではその中身はどう作られたのか。LLMの学習は、大きく3段階に分かれる
 - **事前学習（Pre-training）**：インターネット規模の大量の文章から、直前までのトークンをもとに「次のトークン」を当てさせる問題を、何十億回も繰り返す。これにより文法・知識・ある程度の論理が身につく
 - **指示チューニング（SFT）**：「指示」と「望ましい回答」のペアを使って、質問に答える・指示に従うといった対話の振る舞いを学習させる
 - **アラインメント（RLHFなど）**：人間の評価をもとに、より好まれる・安全で役立つ回答を選びやすくなるよう、さらに調整する
- 事前学習だけでは「文章の続きを当てるだけの機械」。指示チューニングとアラインメントを経ることで、初めて「質問に答える」「指示に従う」というChatGPTのような振る舞いが身につく
- どの段階でも基本の考え方は同じ：モデルの予測と正解（または人間の評価）を比較し、外れていた分だけ内部のパラメータ（数十億～数兆個）を少しずつ調整する



関連用語

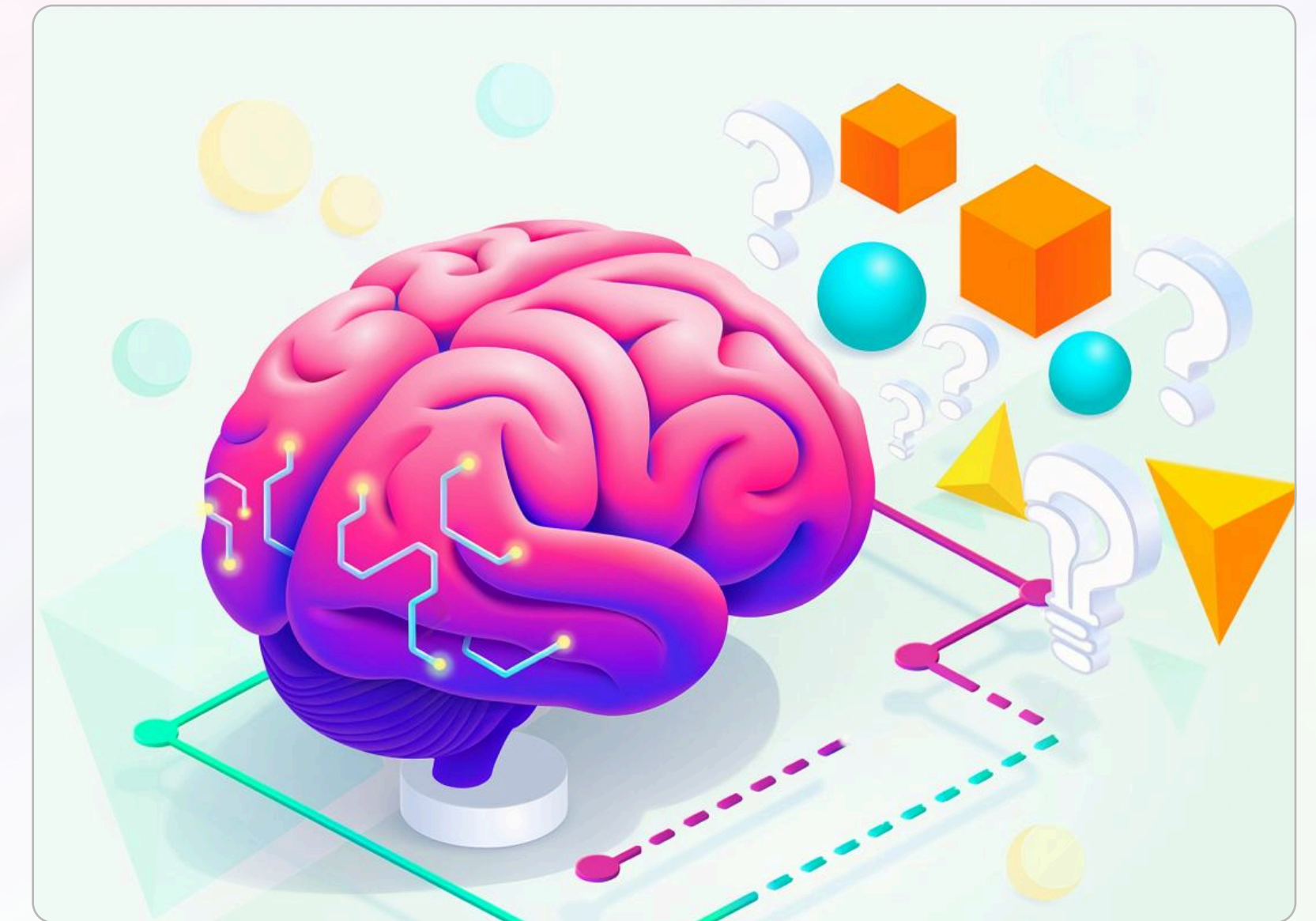
事前学習（Pre-training）

指示チューニング（SFT）

アラインメント（RLHF）

なぜ「推論」できるように見えるのか

- 「次の一言を当てるだけ」のはずなのに、なぜ筋道立った説明や計算ができるように見える？
- 大量の文章には、単なる事実だけでなく「筋道を立てて考える過程」そのものが大量に含まれている → 次トークン予測を極限まで繰り返すことで、そうした「考える過程」のパターンまで内在化されていく
- さらに、途中の考え方を**自分で一度出力し、それを次の入力（コンテキスト）として使う**と、複雑な問題ほど精度が上がるということが知られている
- 例：「まず何算をするか考えて、それから計算する」と一度書き出させると、いきなり答えを出させるより正確になりやすい
- これは**Chain-of-Thought（思考の連鎖）**と呼ばれる手法で、モデル自身の出力を新たな手がかり（コンテキスト）として使っている
- つまり「推論」に見える挙動は、複雑な内部の思考回路というより、**自分の出力を含めた文脈をうまく使って、次の一言を当て続けた結果**、という側面が大きい



関連用語

Chain-of-Thought

LLMの限界

- LLMは、知っているのではなく予測している
- LLMの中核（学習済みモデルそのもの）は「事実を検索して答えている」のではなく、「**学習データから、もっともらしい続きを予測している**」だけ
- そのため、自信満々に事実と異なる内容を生成してしまうことがある
- これを**ハルシネーション（幻覚）**と呼ぶ
- 例：存在しない論文のタイトルや著者名を、いかにも本物らしく作り出してしまう
- 「もっともらしさ」と「正しさ」は別物、という前提を持つことが重要

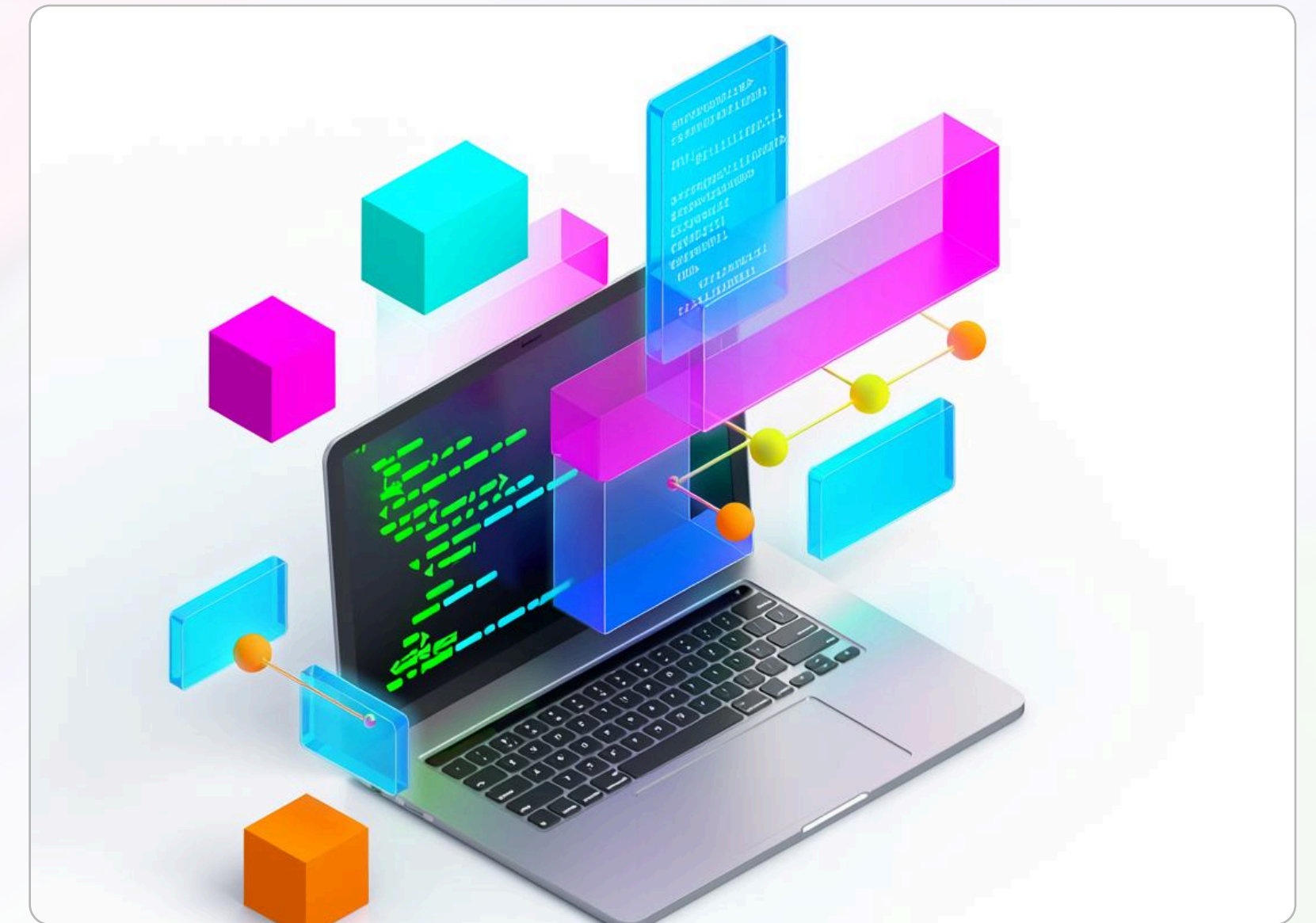


関連用語

ハルシネーション

コンテキストエンジニアリングへ

- ここまでの内容から分かること：LLMの出力は、**学習で身につけたパラメータ**と、そのとき**入力として渡された文脈**の両方をもとに、次のトークンを予測した結果
- パラメータは学習が終わった時点で固定されているが、**文脈（コンテキスト）は、毎回自由に渡し方を変えられる**
- つまり、**同じ質問でも、渡す文脈（前提・指示・参考情報）が変われば、答えの質は大きく変わる**
- **良い前提・具体的な指示・必要な情報を渡すほど、確率分布の中で「望ましい答え」の確率が高まりやすくなる**
- - この「LLMに何を、どう渡すか」を設計する技術を**コンテキストエンジニアリング**と呼ぶ
- 次回は、この「渡し方」を具体的に扱っていく



関連用語

コンテキストエンジニアリング

LLMの内部の仕組みを理解できましたか？

気になるスライドがあったら、それを深掘りしてみてください。
深掘りすることで、他の分野へと繋がり、世界がさらに広がってゆきます。

SYRUP

<https://syrup.papipop.jp>

Thank You!